Documentation

Amin Haidar, Samuel Pearce, Julian Werner

1. Table of contents

2. Project Description

For our project we originally had a text-based strategy game, to be run in the console in mind. This proved to be too exciting of a prospect, and we were unable to restrain ourselves from adding too many cool features. This idea was shelved for another time.

Our design pivoted to a simpler text-based game like Zork[1]. We discussed what the story should be and settled on a game where you have to escape from TBZ. Each level is a floor and each floor gets you one step closer to the exit on the ground floor. Then some doors will be locked and you must look around the place to find keys and other items as well as talk to NPCs (Non Player Characters). Our next thought was to give the player a "hunger" and "thirst" meter. If either one reaches zero, the player dies and must start the level again. The meters are replenished by eating and drinking assorted food the player finds around TBZ. Every "Action" the player makes (moving and picking things up), causes the hunger meter to drop by one and the thirst meter drops by two.

[1] http://textadventures.co.uk/games/play/5zyoqrsugeopel3ffhz_vq

2.1. File Format

The level structures can also be loaded from files, meaning that the game can be expanded upon later if required. The file format is as follows:

Each line is a single property with a name, an equals symbol (=) and then the content. E.g.:

```
property.name = Hello, world!
```

Each file corresponds to one level in the game and each file holds every room and every item, door and interactable object in that room. Here's an example of a full room description with comments: (The comments begin with a hash #)

```
    #Every room's name must be 'rm' plus a number (0-50)
    #The first attribute is the room's description
    rm1 = An empty hallway with some shoes by the door

    #Going north from this room takes you to 'rm2'
    rm1.north = rm2

    #This creates a door to the north named 'Main Door'
    #This door must be opened with an item named 'Main Key'
    rm1.door.north = Main Door
    rm1.door.north.key = Main Key

    #Optionally you can set whether the door is locked by default
    rm1.door.north.locked = false

    #Optionally you can also set what message the door returns
    #when it is opened, can't be opened or when the player
    #interacts with it
    rm1.door.north.succmsg = The door creaks open slowly.
    rm1.door.north.failmsg = The door doesn't budge.
    rm1.door.north.interactmsg = This is a large oak door.

    #This creates an item in the current room named 'Main Key'
    #Every item name must be 'i' plus a number (0-5)
    rm1.i1 = Main Key

    #Optionally you can also set how much hunger and thirst
    #the item replenishes when consumed
```

```
    rm1.i2 = Bottle of Water
    rm1.i2.hunger = 0
    rm1.i2.thirst = 20

    #You must also tell the game where the player starts and
    #which room they must reach to finish the level
    startroom = rm1
    endroom = rm2

    #This creates an interactable in the current room named 'Teacher'
    #Every interactable name must be 'int' plus a number (0-5)
    rm1.npc1 = Teacher

    #This defines the default interaction with the object
    #You can optionally set a command to be run when the
    #object is interacted with.
    rm1.npc1.def.msg = Do you want the rest of this sandwich?
    rm1.npc1.def.cmd = hunger(10)

    #The item interaction can be optionally defined
    #Otherwise the response will simply be "You can't use that with
that"
    rm1.npc1.itm = Doctor Note
    rm1.npc1.itm.msg = Oh, here's the elevator key. Get well soon.
    rm1.npc1.itm.cmd = give(Elevator Key)
```

3. Milestones

3.1. Project Planning

The first milestone will be reached upon having created all UML diagrams and the test-cases. This milestone marks the beginning of practical progress. Before this point no code will have been written yet. Although, after this milestone, some elements of the diagrams may change in accordance with the projects progress.

3.2. Classes Created and Tested

This milestone will be reached once all classes have been written, have their full functionality and have been unit-tested. This milestone means the bulk of the project is complete. Only the integration testing and bug-fixing is left of practical work on the project.

3.3. Testing Complete

This milestone will be reached when all the integration tests are completed. This means that the application itself is complete and fully functional. At this point the only remaining work is finalising the documentation.

3.4. Project Complete

This milestone will be reached after all previous milestones have been reached and the project`s documentation is complete and accurate. The only work after this milestone is popping the champagne.

4. Dependencies

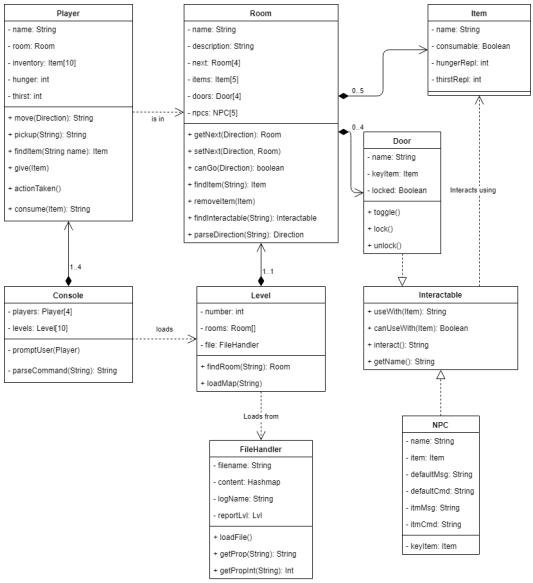
4.1. SimpleTUI - v2.0.2

GitHub Page[2]

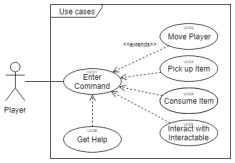
This library wouldn’t have been necessary, but it saves us a lot of time on boring and boilerplate input validation. Not to mention: The library itself was made by Sam previously. Therefore, no code was actually taken from any third-parties for the input validation.

[2] https://github.com/Sam36502/SimpleTUI

5. UML Class Diagram



6. UML Use Case Diagram



7. Use Cases

UC01 - Enter Command

Actor

Player

Basic Flow

The player enters text into the console and presses the carriage return key. If the command matches a command in the system, then the command is executed. If the command doesn't match any known command, then the player is informed, that this command wasn't recognised.

If the action made the player perform something, i.e `move`, `use` or `pick-up`, then two points are taken off their "thirst" meter and one point is taken from their "hunger" meter.

Preconditions

The game has been started and the player has made it past the main-menu.

Termination

The player has performed an action in the game or are informed, why this could not be performed.

UC02 - Move Player

Actor

Player

Basic Flow

The player enters the `move` or `mv` command into the console with a single argument that is either `n`, `north`, `w`, `west`, `e`, `east`, `s` or `south`. If the syntax is correct and there exists an open path between the current room and the room in the direction provided, then the player will be moved to the room in the given direction. If such a movement is not possible, then the user is informed that they are unable to move in that direction.

Preconditions

The player must be in a level and in a room with an open path to an adjacent room.

Termination

The player is now in the room in the direction that they selected from the first room.

UC03 - Pick up Item

Actor

Player

Basic Flow

The player enters the `pick-up` or `pu` command into the console with a single argument. This argument must be the name of an item in the room. If the named item exists in the room, then it is removed from the room and put into the player`s inventory. If the Item cannot be found in the current room, the player is informed, that no such item exists.

Preconditions

The player must be in a room that has at least one item in it.

Termination

The item is no longer present in the room and has been added to the user`s inventory.

UC04 - Consume Item

Actor

Player

Basic Flow

The player enters the `eat`, `drink` or `cs` command into the console with a single argument. This argument must be the name of a consumable item in the player`s inventory. If the player has this item and the item is

consumable, then the item is consumed and the user`s "thirst", "hunger" and "urine" meters are updated accordingly. If the user does not posess the named item, then they are informed thusly. If the item they do name is owned, but not consumable, then they are told, they consume the given item.

Preconditions

The user has a consumable item in their inventory.

Termination

The item is removed from the user’s inventory and their necessity meters are updated accordingly.

UC05 - Interact with Interactable

Actor

Player

Basic Flow

The player enters the `use`, `interact`, `uo` or the `in` command into the console with two arguments. The format of the command is:

```
use <item> on/with <interactable object
```

If the player has the named item in their inventory and the interactable object exists inside the current room, then the interaction takes place. Otherwise, the player is simply informed that they are unable to perform this interaction.

Preconditions

The player is in a room with at least one interactable object and have at least one item in their inventory.

Termination

The item is no longer in the user`s inventory and the interaction takes place.

UC06 - Get Help

Actor

Player

Basic Flow

The user enters the `help` or `?` command into the console. Upon executing this command, the console will display a list of all currently available commands as well as a list of their necessary arguments. If the user enters the command with an argument, a more in-depth explaination is outputted along with some examples of how to use that specific command.

Preconditions

The user must be able to enter commands and be at least past the main menu.

Termination

The user is given the information they requested.

8. Unit Test Cases

8.1. Item

Table 1. Item Test Cases

Input	Expected	Output	Approval
Item("")	ItemNameExc.	ItemNameExc.	OK
Item(" ")	ItemNameExc.	ItemNameExc.	OK
Item("123")	ItemNameExc.	ItemNameExc.	OK
Item("abc")	-	-	OK
Item("", 1, 1)	ItemNameExc.	ItemNameExc.	OK
Item("abc", -1, -1)	ItemReplExc.	ItemReplExc.	OK
Item("abc", 101, 101)	ItemReplExc.	ItemReplExc.	OK
Item("abc", 0, 100)	-	-	OK

8.2. Door

Table 2. Door Test Cases

Input	Expected	Output	Approval
Door("a", Item("key"))	-	-	OK
canUseWith(Item("a", 1, 1))	false	false	OK
canUseWith(Item("ath"))	true	true	OK
isLocked()	true	true	OK
useWith(Item("abc"))	"That won’t work…"	"That won’t work…"	OK
useWith(Item("key"))	"The key clicks…"	"The key clicks…"	OK
isLocked()	false	false	OK

8.3. Room

Table 3. Room Test Cases

Input	Expected	Output	Approval
rm1=Room("rm1", "A room.")	-	-	OK
rm2=Room("rm2", "Another room")	-	-	OK
getOpposite(WEST)	EAST	EAST	OK
parseDirection("North")	NORTH	NORTH	OK
parseDirection("w")	WEST	WEST	OK
rm1.setNext(NORTH, rm2)	-	-	OK
rm1.getNext(NORTH)	rm2	rm2	OK
rm1.canGo(NORTH)	true	true	OK
rm2.canGo(SOUTH)	false	false	OK
rm1.setDoor(NORTH, Door("d", Item("key")))	-	-	OK
rm1.canGo(NORTH)	false	false	OK

8.4. FileHandler

(Shortened to FH for convenience)

For the purposes of this test, it is assumed, that there is a file named "test.txt" in an adjacent folder to the executable. The file "nonexistent.txt" does not exist.

The contents of "test.txt"

```
#This is a comment and will be ignored
This is invalid and will be ignored
prop1=Hello! #End-of-line comment
prop2 = 42
```

Table 4. FileHandler Test Cases

Input	Expected	Output	Approval
fherr=FH("nonexistent.txt")	-	-	OK
fh=FH("test.txt", "test", WARNING)	-	-	OK
fh.getFilename()	test.txt	test.txt	OK
fh.loadFile()	-	-	OK
fh.getProp("prop1")	Hello!	Hello!	OK
fh.getProp("prop2")	42	42	OK
fh.hasProp("prop1")	true	true	OK
fh.getPropInt("prop1")	[WARNING]…	[WARNING]…	OK
fh.getPropInt("prop2")	42	42	OK
fherr.getProp("prop1")	InvalidPropExc.	InvalidPropExc.	OK
fherr.hasProp("prop1")	false	false	OK
fherr.loadFile()	(Sys Exit)	(Sys Exit)	OK

8.5. Level

This unit test will mainly return empty values. To test how it works in conjunction with the other classes, see the "Level Loading" Integration test

Table 5. Level Test Cases

Input	Expected	Output	Approval
Level(0)	LevelNumberExc.	LevelNumberExc.	OK
Level(26)	LevelNumberExc.	LevelNumberExc.	OK
Level(1)	-	-	OK
lvl.findRoom("nonexistent")	RoomNameExc.	RoomNameExc.	OK
lvl.loadMap("nonexistent")	(Sys Exit)	(Sys Exit)	OK
lvl.getNumber()	1	1	OK
lvl.getRooms()	(Empty Array)	(Empty Array)	OK

8.6. Player

Table 6. Player Test Cases

Input	Expected	Output	Approval
p=Player("test", rm1)	-	-	OK
p.getRoom()	rm1	rm1	OK
p.getName()	test	test	OK
p.getHunger()	100	100	OK
p.getThirst()	100	100	OK
p.isAlive()	true	true	OK
p.give(Item("water", 5, 5))	-	-	OK
p.findItem("WaTeR")	water	water	OK
p.findItem("abc")	null	null	OK
p.consume(Item("abc"))	You don’t have…	You don’t have…	OK
p.consume(Item("water", 5, 5))	You had the water…	You had the water…	OK
p.getHunger()	100	100	OK
p.getThirst()	100	100	OK

8.7. Console

Table 7. Consle Test Cases

Input	Expected	Output	Approval
help	List of commands	List of Commands	OK
help mv	Information about mv	Information about mv	OK
look abc	Look needs no args	Look needs no args	OK
abcdef	Invalid Command	Invalid Command	OK

9. Integration Test Cases

These tests assume a text file named "test.txt" exists adjacent to the current executable and has following contents:

```
#Integration Testing level

# Level Attributes #
startroom = rm1
endroom = rm2

# First Room #
rm1 = An empty hallway with some shoes by the door

rm1.north = rm2

rm1.door.north = Main Door
rm1.door.north.key = Main Key

rm1.door.north.locked = true

rm1.door.north.succmsg = The door creaks open slowly.
rm1.door.north.failmsg = The door doesn't budge.
rm1.door.north.interactmsg = This is a large oak door.

rm1.i1 = Main Key

rm1.i2 = Bottle of Water
rm1.i2.hunger = 0
rm1.i2.thirst = 5

# Second Room #
rm2 = You reach a stairwell to the next floor down and go down it.
```

9.1. Level Loading Test

Table 8. Level Loading Test Cases

Input	Expected	Output	Approval
lvl=Level(1)	-	-	OK
lvl.loadMap("test.txt")	-	-	OK

Input	Expected	Output	Approval
lvl.getRooms()	-	-	OK
rm1.getName()	rm1	rm1	OK
rm1.canGo(NORTH)	false	false	OK
rm1.findItem(Main Key)	Main Key	Main Key	OK
rm1.findItem(Bottle of Water)	Bottle of Water	Bottle of Water	OK
rm1.getDoors()	-	-	OK
door.interact()	This is a large…	This is a large…	OK
door.useWith(Item(Main Key))	The door creaks…	The door creaks…	OK
rm1.canGo(NORTH)	true	true	OK

9.2. Player Interaction Test

Table 9. Player Interaction Test Cases

Input	Expected	Output	Approval
l=Level(1)	-	-	OK
l.loadMap("test.txt)	-	-	OK
p=Player("test", rm1)	-	-	OK
p.getHunger()	100	100	OK
p.getThirst()	100	100	OK
p.move(NORTH)	You can’t go…	You can’t go…	OK
p.pickup("abc")	You can’t see…	You can’t see…	OK
p.pickup("Bottle of Water")	The Bottle of…	The Bottle of…	OK
p.pickup("Main Key")	The Main Key was…	The Main Key was…	OK
p.getHunger()	98	98	OK

Input	Expected	Output	Approval
p.getThirst()	96	96	OK
north door.useWith(Main Key)	The door creaks…	The door creaks…	OK
p.move(NORTH)	You reach a…	You reach a…	OK
p.consume(Bottle of Water)	You had the…	You had the…	OK
p.getHunger()	97	97	OK
p.getThirst()	100	100	OK

9.3. Console full Integration Test

For the full integration test the following text document was used:

```
# Last Level

# Level Attributes
startroom = rm0
endroom = rm4

# First Room
rm0 = You're in an eerie Classroom with nobody inside. A door leads out
 to the north.

rm0.door.north = Metal Door
rm0.door.north.key = Small Key
rm0.door.north.succmsg = The door swings open swiftly.
rm0.door.north.failmsg = The door lets out a metallic thud, but doesn't
 move.
rm0.door.north.interactmsg = It's a robust metal door. Forcing it
 wouldn't work.

rm0.i0 = Small Key

rm0.i1 = Sandwich
rm0.i1.hunger = 10
rm0.i1.thirst = 0
```

```
# Connection
rm0.north = rm1
rm1.south = rm0

# Second Room
rm1 = You're in an airy and bright hallway continuing to the east and
 west.

# Connections
rm1.west = rm2
rm1.east = rm3
rm2.east = rm1
rm3.west = rm1

# Third Room
rm2 = You're at the end of the hallway.
rm2.i0 = Rusty Key

# Fourth Room
rm3 = You're at the end of the hallway. You can see some stairs through
 glass doors to the south.
rm3.door.south = Glass Door
rm3.door.south.key = Rusty Key

# Connection
rm3.south = rm4

# Exit Room
rm4 = You see a set of stairs heading down and continue down to the next
 level...
```

Table 10. Console full Integration Test Cases

Input	Expected	Output	Approval
move north	Can’t see that	Can’t see that	OK
look	Room description	Room description	OK
pickup abc	Can’t see that	Can’t see that	OK
pickup Small Key	Picked up	Picked up	OK
pickup Sandwich	Picked up	Picked up	OK
eat abc	Don’t have that	Don’t have that	OK

Input	Expected	Output	Approval
eat Sandwich	Eaten	Eaten	OK
interact-with abc	Can’t see that	Can’t see that	OK
interact-with Metal Door	Door description	Door description	OK
use abc on abc	Don’t have that	Don’t have that	OK
use Small Key on abc	Can’t see that	Can’t see that	OK
use Small Key on Metal door	Door opens	Door opens	OK
go north	Room Description	Room Description	OK
go west	Room Description	Room description	OK
pickup Rusty Key	Picked up	Picked up	OK
inventory	List of items	List of items	OK
go east (x2)	Room Description	Room Description	OK

10. Work Journal - Part A

10.1. Day - 1

1/13/2019

We began by Setting up our documentation method. In our case, we decide to use Asciidoc, as we'd rather not spend our time styling the documentation, when it can all be automated with some basic tools. Not to mention the single simple text format can be exportet to many different document types such as PDF, HTML and DocBook. This means we would also have the option of hosting our documentation as a website.

Next we began making the UML diagrams. We both discussed the class structure and what we would need to store the level structure and item information. Once again, we had many good ideas to add, but reeled in our ambitions for the first version and decided to add the new features later in a version 2.0 if we got the time.

Once we had the class diagram, we split up to work on the use-case and sequence diagrams. Amin began writing the use-cases and created a diagram, that we both agreed, protrayed our project faithfully. (Although both our experience with UML is rather limited)

Sam then did some research on sequence diagrams and got to work creating them. At first we weren't quite sure how to visualise conditionality in the diagram, but we eventually got the hang of using alternate paths.

Finally, we were finished with most of the planning stage and planned to start programming next time.

10.2. Day - 2

13/10/2019

Over the weekend we both chose some Classes to create and individually unit-test. We occasionally had some issues and had to discuss how the files should

be loaded. However, we were able to standardise our system so that if we wanted to add any more new features, then it is easily expandable.

Amin took the Item, Interactable, Door and Room classes and implemented and tested them. Sam took on the Level and FileHandler classes. These were not as simple as the others. The level class contained one rather monolithic function for loading all the world attributes into the rooms from a file.

Due to the Item class being created first it was documented far more in-depth than the other ones. it was documented the most in-depth. We discussed writing a seperate documention for each class in order to more completely document each part of the game.

Sadly, the time is lacking, due to our early slacking in the project and we will keep the shortened documentation. We will always have time to add the improved documentation later.

We hope to finish off the remaining classes during our free time over the next week.

10.3. Day - 3

16/10/2019

During the week we had each completed our parts of the classes. We managed to create and unit test all of them. We also performed the integration tests for all the classes to make sure they would all work together in unison.

We were both surprised to see how effective this method of development was. Previously, I probably would have started with the Console class, but now I see it makes more sense to form the OOP model and thoroughly test it before implementing the main functionality.

After creating and testing the Console class working in perfect harmony with all the other classes was a sight to behold for junior programmers such as ourselves. After that all that was left for programming was to make sure that each player was given a set number of turns per round and that each player could play until the

end of level. We also added a leaderboard, so that all the players can see who finished first, second, third and etc.

We have both decided to update a few parts of the documentation and generally polish the finished product, while also creating a world based on our original story idea. Now the engine is complete, we can work on the actual game.

10.4. Day - 4

22/10/2019

Once the engine was complete we decided to start designing the levels and planning for extra features we could add in. Sam began implementing the generic interactbable object functionality, while Amin planned the first three levels.

Sam also set up a small website to navigate the documentation. It took us a while to get the the documentation standardised. Sam had been writing most of the documentation and hadn't properly understood how the unit-tests are supposed to work. Luckily we noticed this during the lesson and were able to remedy the situation. Next we decided to finish what we were working on over the week so the project should be mostly done by the next tuesday.

11. Work Journal - Part B

11.1. Day - 1

03/12/2019

For the second half of the module, Julian joined our team and we began discussing what project we would like to work on. At first we were going to implement a game that Amin and Sam had already been planning for a while. We began by describing the vision of the project to Julian to catch him up and we collected some requirements based on the ÜK we were having at the same time. Sam and Julian were being taught about Requirements Engineering at NOSER.

We decided that now we had nailed down the vision, we would continue adding requirements to the list over the course of the next week.

11.2. Day - 2

10/12/2019

Well, we found it's quite difficult to think of requirements by themselves. We probably should have used one of the many techniques of RE such as an interview or brainstorming session. We decided that the scope of our project might be a bit too large for the given amount of time. We settled on the idea of simply extending and continuing work on the text-adventure game engine. We all collectively agreed this would be far more manageable. Immediately we came up with some — once again, overambitious — ideas. It became clear that we needed a more concrete definition of what we needed to do. Not to mention Julian hadn't even used the engine before. Amin and Sam promptly gave him a chance to play the game. Many small bugs we had previously ignored or not even noticed became clear quite quickly. We decided, these should be fixed and the rest of the game should also be made. In a project board, we made concrete user-stories and bugs to fix and implement.

11.3. Day - 3

17/12/2019

Amin decided to begin working on a script to help create the levels. This way we would be able to finish the game. Julian said he would do research on how we could make a REST API to interact with the game. Sam would avise both on how to achieve these goals; he would act as project manager.

Amin soon got a small prototype of the level designer working, and we eagerly designed some test levels and let each other try playing them. An interesting feature of the engine is that it doensn't follow a fixed grid for the world layout. This means that you can create non-euclidean levels that make no sense. Still, we decided it would probably be best if the player didn't get lost easily. I wonder if Mr. Käser will even bother reading this far…

Julian and Sam discussed how it would be possible to make an API that allows the users to play the game in a browser window without even having to install the

program or Java. Given the game is only a text-based game, you could send all the commands to an HTTP GET endpoint and return a string of what happened. Then creating a website with an interactive terminal would be a trivial matter.

11.4. Day - 4

07/12/2019

The API plan fell through; it didn't take into account that the whole program would need to be rebuilt on a new Springboot foundation. There would only be one more week of work, and we would also have to study for some tests; it simply wasn't practical. We focussed our efforts on the level designer script and got the full version working within a week. Amin had handled most of the logic and file generation. Sam handled the input validation. One shouldn't spoil the ship for a ha'porth of tar! If you're going to do something, might as well do it properly.

Then we all began reviewing and designing levels for the full game. The whole development process would be greatly accelarated by the convenience of the script. Julian had also been working on making some of the outputs less ambiguous, while fixing the left-over bugs simultaneously.

11.5. Day - 5

21/12/2019

The final week was mostly spent implementing the levels we designed earlier and polishing any unseemly issues. Amin was head of level design. He had done most of the designs. Julian was mostly testing to make sure the game worked correctly. Sam was taking the levels and implementing them with Amin's script. Soon enough, we had 10 at least somewhat interesting levels. We compiled all the necessary files into a zip archive and pushed our release to the GitHub repository. We also created an installer that would put all the necessary files where they needed to go. This would make it very easy to go straight from download to playing the game.

While it certainly has its issues, and Sam has even planned to make a completely new version from the ground up, we are all quite proud and happy to be finished with this project.